

A TOOLBOX FOR H_{∞} OPTIMIZATION

NAGW-1333

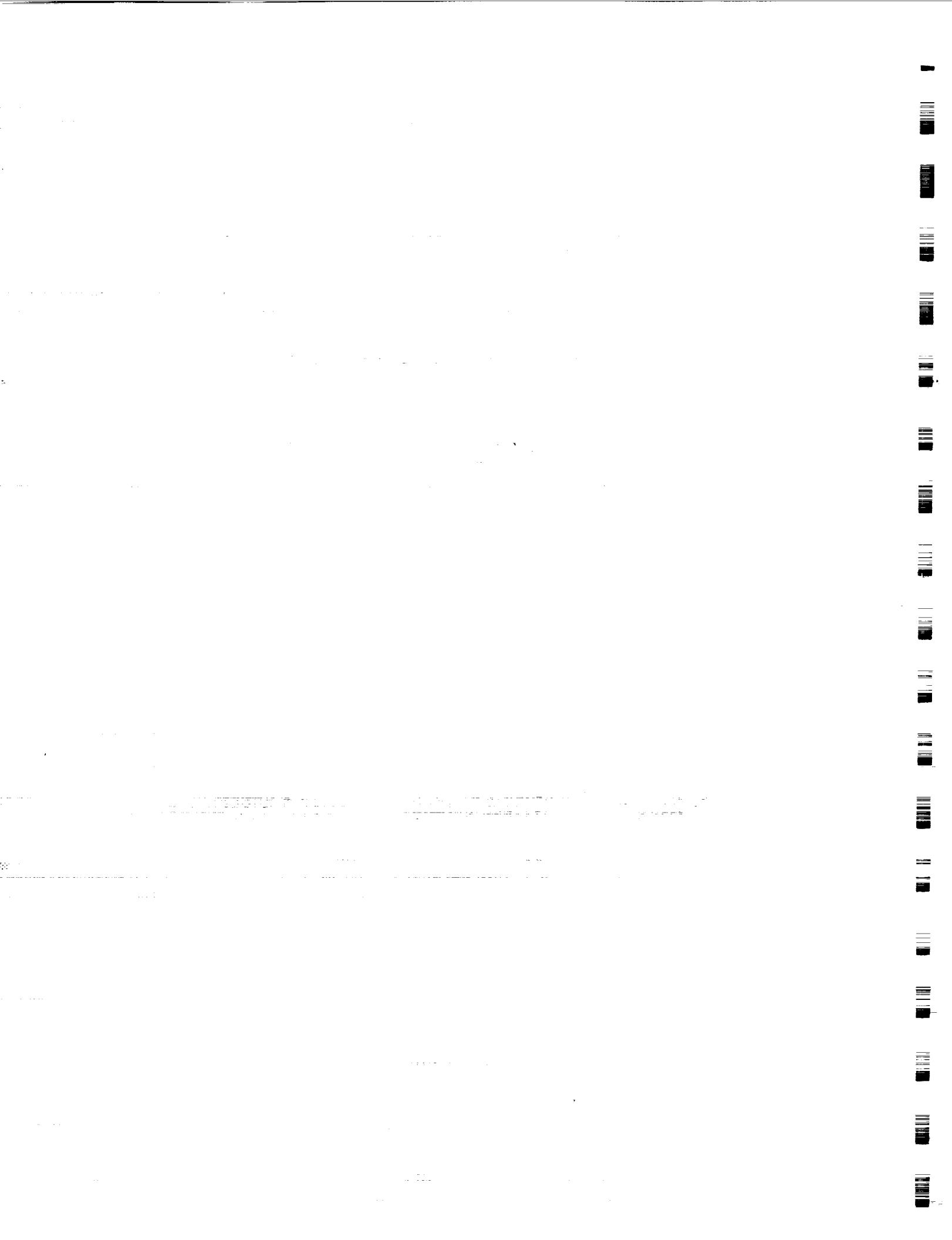
By:

**K. Balachandran
D. Sood
J. Wen
J. Chow**

**Department of Electrical, Computer and Systems Engineering
Department of Mechanical Engineering, Aeronautical
Engineering & Mechanics
Rensselaer Polytechnic Institute
Troy, New York 12180-3590**

December 1988

CIRSSE Document #41



A Toolbox for H_∞ Optimization

Kumar Balachandran, Deepak Sood, John Wen and Joe Chow

ECSE Department

Rensselaer Polytechnic Institute

December 1988

Abstract

The use of stable factorization provides a simple tool for the design of control systems to meet the frequency domain specifications. This report describes the development of software to design optimal controllers through stable factorization. The object of this exercise is to develop a package that can be used to solve the classical H_∞ optimization problem.

The implementation of the toolbox is carried out in Pro-Matlab.

Contents

1	Introduction	3
2	Data Structure	4
3	The Algorithm	4
4	Conclusions	7
5	Appendix - List of Routines	9
6	Appendix - Source Code	18
7	Examples	59

1 Introduction

The classical H_∞ optimization problem is stated as follows:

$$\alpha = \inf \{ \| T_1 - T_2 Q T_3 \|_\infty : Q \in RH_\infty \} \quad (1)$$

A matrix Q in RH_∞ satisfying the above lower bound will be called optimal as pointed out by Francis [1].

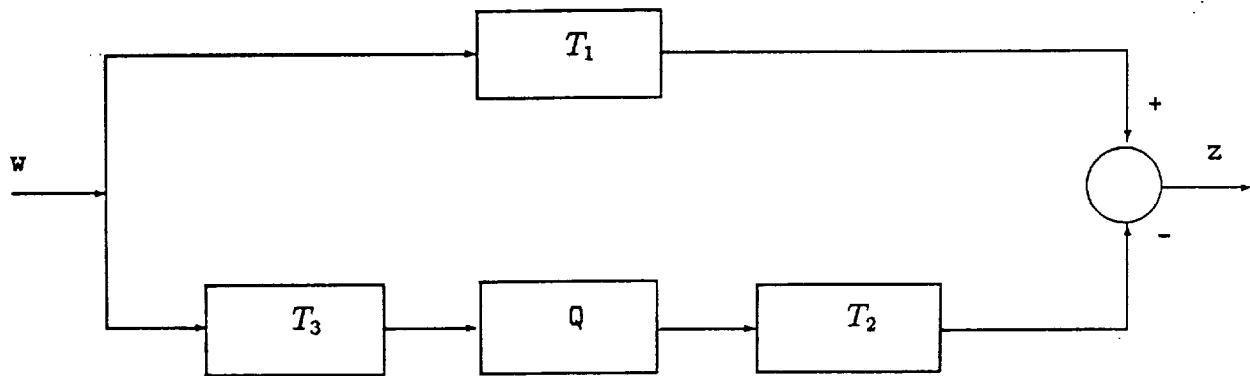


Figure 1. The Model-matching Problem

Our objective is to implement an algorithm to find out this optimal Q given the stable transfer matrices, T_1 , T_2 , and T_3 . It is well known that the stable controller design problem may be easily transformed into the above mentioned model matching problem.

2 Data Structure

The data structure that has been used corresponds to the Transfer Matrix definition give in Francis [1].

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}, nt \quad (2)$$

T represents the transfer matrix and nt represents the number of states of the system.

Most of the routines take T and nt as parameters and separate the system into A , B , C and D matrices. A description of the routines is provided in the appendix attached.

3 The Algorithm

The algorithm used for the solution of the H_∞ optimization problem is described below, [1].

1. Let

$$T_2 = U_i U_o, U_i \text{ inner, } U_o \text{ outer,}$$

and

$$Y := (I - U_i \tilde{U}_i) T_1$$

Compute $\| Y \|_\infty$.

2. Let

$$\alpha_1 = \| T_1 \|_\infty \text{ be an upper bound for } \alpha.$$

3. $\gamma_0 = \| Y \|_\infty$.

Set interval $J = (\gamma_0, \alpha_1]$, $i = 1$.

4. Select a value of $\gamma_i = (\gamma_{i-1} + \alpha_1)/2$.

5. Y_0 = spectral factor of $(\gamma_i^2 - \tilde{Y}\tilde{Y})$

$T_3 Y_0^{-1} = V_{co} V_{ci}$), co-outer and co-inner respectively.

6. $Z = \tilde{U}_i T_1 Y_0^{-1} (I - \tilde{V}_{ci} V_{ci})$

Compute $\| Z \|_\infty$.

7. If $\| Z \|_\infty < 1$, continue;

If not, $\gamma_i = (\gamma_i + \alpha_1)/2$,

return to Step 5.

8. Z_{co} = co-spectral factor of $(I - Z\tilde{Z})$.

9. $R = Z_{co}^{-1} \tilde{U}_i T_1 Y_0^{-1} \tilde{V}_{ci}$.

Compute $\| \Gamma_R \|$.

If $\| \Gamma_R \| < 1$ then

$\alpha_1 = \gamma_i$, $i = i + 1$;

else

$J = (\gamma_i, \alpha_1]$, $i = i + 1$;

If $\alpha_1 - \gamma_i < \epsilon$, set $\alpha = \alpha_1$,

recompute R and continue;

else return to step 4.

10. Find the strictly anti-stable part of R, R_1 .

11. Solve the Lyapunov equations for the observability and controllability Grammians of R_1 ;

$$AL_c + L_c A^T = BB^T$$

$$A^T L_o + L_o A = C^T C;$$

Set $N = (I - L_o L_c)^{-1}$.

$$L_2 = \begin{bmatrix} A & N^T B \\ C & 0 \end{bmatrix},$$

$$L_4 = \begin{bmatrix} -A^T & NL_o B \\ B^T & I \end{bmatrix}.$$

12. $X = R - (L_1 Y + L_2)(L_3 Y + L_4)^{-1}$,

where X is the solution to the classical Nehari problem. Y is a parameterizing stable factor and is assumed to be zero in the implementation.

13. Solve $Q = (Z_{co}^{-1} U_0)^{-1} X V_{co}^{-1}$.

4 Conclusions

In this report, a toolbox has been developed that can be used to carry out stable factor based design of control systems. The appendix contains a list of all the routines and their source code. The Matlab Help facility can be used to obtain information about the usage of these routines. A number of examples from Francis'[1] book were solved successfully using this toolbox. The main feature of this exercise was the solution of the matrix valued H_∞ optimization problem. Future work will involve the design of controllers using the passivity criterion, [2]. Also work will be carried out for the design of reduced dimensional controllers for high (infinite) dimensional systems.

References

- [1] Francis, B.A., "A course in H_∞ control theory," Springer-Verlag, 1987.
- [2] Wen, J.T., "Controller synthesis for infinite dimensional systems based on a passivity approach."
- [3] Doyle, J., et. al." State-space solutions to standard H_2 and H_∞ control problems."

5 Appendix - List of Routines

1. help checknorm

This routine checks whether the infinity norm of a transfer matrix $\| 1/\gamma * G(s) \|$ is less than 1. If so, it returns 1; if not, it returns 0. For further info see Doyle et al., "State space solutions to standard H_2 and H_∞ problems [3]."

USAGE : [truth]=checknorm(gamma,T,nt)

2. help convtf2ss

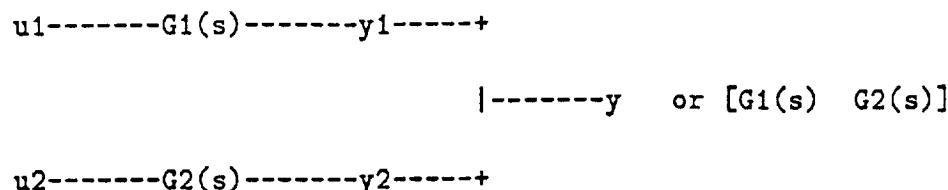
This function converts a MIMO system transfer matrix to the state space domain. The input to this function is the number of rows and columns of the transfer function matrix.

Note that the user has to feed in the coefficients of the numerator and those of the common denominator for each column of the Transfer function matrix i.e. each input to the system.

USAGE : [tcom,ntcom]=convtf2ss(row,col)

3. help fat

Gives the equivalent of a fat connection of systems T_1 and T_2



4. help findlyap

This function solves the Lyapunov equations to find the value of L_c and L_o .

USAGE : [lc,lo]=findlyap(t,nt)

5. help hanksingval

This function calculates the distance from a matrix to H_∞ . First the strictly proper antistable part of the matrix is determined using the function 'parts'.

USAGE : hankval=hanksingval(t,nt)

6. help hinfinorm

This function computes the infinity norm of a transfer matrix. The procedure is iterative and follows the method shown in Doyle et al [3].

USAGE : [n] = hinfinorm(T,nt)

7. help hinfnorm

This function finds the H_∞ norm of a given transfer function.

USAGE : [hinf]= hinfnorm(t,nt)

8. help invert

This function performs the inversion of a square system.

USAGE : [tinv,ntinv]=invert(t,nt)

9. help invretdiff

This function returns the inverse of the return difference matrix in a unity feedback system.

The result is $\text{inv}(I+G)$.

USAGE : [tre,nret]=invretdiff(t,nt)

10. help iplus

Gives the equivalent of $(\gamma * I) + G$ of a system T_1

The result is $\gamma * I + G(s)$

USAGE : [tiplus,ntiplus]=iplus(gamma,t1,nt1)

11. help lcofac

This function provides the left coprime factorization of the plant given as tp with ntp number of states. The user has to supply the values of f and h. These can be obtained from the program lqfdbk.m.

USAGE : [dl,ndl,nl,nnl]=lcofac(tp,ntp,f,h)

12. help lctrcofac

This function provides the left coprime factorization $vr^{-1} * ur$ of the controller with the plant given as tp with ntp number of states. The user has to supply the values f and h. These can be obtained from the program lqfdbk.m.

USAGE : [vr,nvr,ur,nur]=lctrcofac(tp,ntp,f,h)

13. help lqfdbk

This function is for linear quadratic feedback design of the observer based parameterization of the controller. The plant tp and the number of states ntp are input. It uses the lqr for placing the poles of the system and the observer. It iteratively places the poles of the observer to the left of the closed loop poles of the system.

USAGE : [f,h]=lqfdbk(tp,ntp)

14. help mul

Gives the equivalent of a series connection of systems $T_1 * T_2$

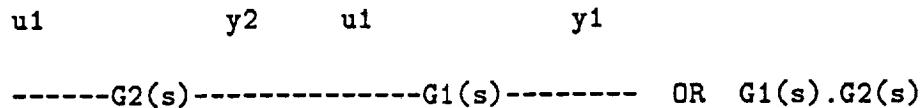
$$\begin{array}{cccc} u_1 & y_2 & u_1 & y_1 \\ \hline \hline G_2(s) & & G_1(s) & \end{array} \quad \text{OR} \quad G_1(s).G_2(s)$$

The transfer function $G_1(s)$ is given by T_1, n_{t1} and $G_2(s)$ by T_2, n_{t2} .

USAGE : [tmul,ntmul]=mul(t1,nt1,t2,nt2)

15. help mul1

Gives the equivalent of a series connection of systems $T_1 * T_2$



The transfer function $G_1(s)$ is given by $T_{1,nt1}$ and $G_2(s)$ by $T_{2,nt2}$.

USAGE : [tmul,ntmul]=mul(t1,nt1,t2,nt2)

16. help negate

Gives the equivalent of negation of a system T_1 .

The result is $-G(s)$

USAGE : [tneg,ntneg]=negate(t1,nt1)

17. help nehari

This routine will solve the nehari problem as applied to controller design by minimizing the function:

$$\| T_1 - T_2 * Q * T_3 \|$$

The function returns the value of Q which when used in the standard parameterization, yields the controller needed to stabilize a plant.

USAGE : [Q,nq] = nehari(T1,nt1,T2,nt2,T3,nt3)

18. help parallel

Gives the equivalent of a parallel connection of systems T_1 and T_2 .

$$\begin{array}{c} \text{--G1(s)-----y1(s)----+} \\ \text{u---|} \qquad \qquad \qquad |----y \quad \text{or} \quad G1(s)+G2(s) \\ \text{--G2(s)-----y2(s)----+} \end{array}$$

19. help parts

This function breaks the given system into a stable and an antistable part. The function uses the sorteig routine to sort the eigenvalues of the system. The stable and the unstable parts are returned as $T_{1,nt1}$ and $T_{2,nt2}$ respectively.

USAGE : [t1,nt1,t2,nt2]=parts(t,nt)

20. help rcofac

This function provides the right coprime factorization of the plant given as T_p , with ntp number of states. The user has to supply the values of f and h. These can be obtained from the routine lqfdbk.m.

USAGE : [dr,ndr,nr,nnr]=rcofac(tp,ntp,f,h)

21. help rctrcofac

This function provides the right coprime factorization $U_l V_l^{-1}$ of the controller with the plant given as tp with ntp number of states. The user has to supply the values f and h. These can be obtained from the program lqfdbk.m.

USAGE : [vl,nvl,ul,nul]=rctrcofac(tp,ntp,f,h)

22. help smallgain

This function calls the findlyap function to find the values of L_c and L_o . It solves the small gain problem (sector [-1,1] case). Y is assumed to be zero. Refer Francis [1] Pg. 125.

USAGE : [x,nx]=smallgain(r, nr)

23. help sorteig

This function sorts the given system into a stable and an antistable part. The generator matrix of the system returns with eigenvalues in diagonally increasing order. It also returns the diagonalizing transformation v.

USAGE : [ts,nts,v]=sorteig(t,nt)

24. help spectral

This function calculates the spectral factors of a system. The assumption is that the D matrix is invertible. Refer to Page 90 (Section 7.3) of Francis [1].

USAGE : [ts,nts]=spectral(t,nt)

25. help spectrdns

This function calculates the spectral factors of a $\tilde{G}G$. The assumption is that the system has no poles on the jw axis. For the case where D is invertible(square system) see spectral.m.

USAGE : [ts,nts]=spectrdns(t,nt)

26. help systsep

This function separates the transfer matrix into the state space.

USAGE : [a,b,c,d]=systsep(t,nt)

27. help tall

Gives the equivalent of a tall connection of systems T_1 and T_2

--G1(s)-----y1(s)---- -- --
u--| or | G1(s) |
--G2(s)-----y2(s)---- | G2(s) |
-- --

USAGE : [ttall,nttall]=tall(t1,nt1,t2,nt2)

28. help tilde

Complex conjugate transpose

USAGE : [ttilde,ntilde]=tilde(t,nt)

29. help transfer

This function returns the state space analog (T) of a transfer function. nt is the number of state variables.

USAGE : [t,nt]=transfer(a,b,c,d)

30. help transpose

This function finds the transpose of the given system T.

USAGE : [ttrans,nttrans]=transpose(t,nt)

6 Appendix - Source Code

```
1. function [truth] = checknorm(gamma,T,nt)

[a,b,c,d] = systsep(T,nt);

gaminsq = 1/(gamma*gamma);

H = [a gaminsq*b*b';-c'*c -a'];

ei = eig(H);

[row,col] = size(H);

truth = 1;

for i = 1:row,

    if abs(real(ei(i)))<1e-14,

        truth = 0;

        break;

    end

end
```

```

2. function [tcom,ntcom] = convtf2ss(row,col)

for j=1:col;,

    disp('Make sure that you form a common denominator'),

    disp('Input the numerator coeffs. matrix for'),

    disp(j),

    disp('input'),

    num=input(' '),

    disp('Input the denom. vector'),

    den=input(' '),

    [a,b,c,d]=tf2ss(num,den);

    [t,nt]=transfer(a,b,c,d)

    if j==1,

        tcom=t;,

        ntcom=nt;,

    else [tcom,ntcom]=fat(tcom,ntcom,t,nt);,

    end

end

[a,b,c,d] = systsep(tcom,ntcom);

[a,b,c,d] = minreal(a,b,c,d);

[tcom,ntcom] = transfer(a,b,c,d);

```

```
3. function [Tfat,ntfat] = fat(T1,nt1,T2,nt2)

[At1,Bt1,Ct1,Dt1] = systsep(T1,nt1);

[At2,Bt2,Ct2,Dt2] = systsep(T2,nt2);

At = [At1 zeros(nt1,nt2);zeros(nt2,nt1) At2];

[r1,c1]=size(Bt1);

[r2,c2]=size(Bt2);

Bt = [Bt1 zeros(r1,c2);zeros(r2,c1) Bt2];

Ct = [Ct1 Ct2];

Dt = [Dt1 Dt2];

[at,bt,ct,dt] = minreal(At,Bt,Ct,Dt);

[Tfat,ntfat] = transfer(at,bt,ct,dt);
```

4. function [lc,lo] = findlyap(t,nt)

```
[a,b,c,d]=systsep(t,nt);
```

```
lc=lyap(a,-b*b');
```

```
lo=lyap(a',-c'*c);
```

5. function [hankval] = hanksingval(t,nt)

```
[t1,nt1,t2,nt2]=parts(t,nt);

[am,bm,cm,dm]=systsep(t2,nt2);

i=0;

for j=1:nt2,

    t = abs(imag(am(j,j)));

    if t > 1e-6,

        i=i+1;

        k(i)=j;

    end

end

si = i;

u=eye(nt2);

if i ~= 0,

    for j=1:2:i,

        temp=k(j);

        u(temp,temp+1)=1;

        u(temp+1,temp)=-sqrt(-1);

        u(temp+1,temp+1)=sqrt(-1);

    end

end
```

```
ut=u;  
  
am=u*am*inv(u);  
  
bm=u*bm;  
  
cm=cm*inv(u);  
  
lc=lyap(am,-bm*bm');  
  
lo=lyap(am',-cm'*cm);  
  
hankval=sqrt(real(max(eig(lc*lo))));
```

```
6. function [n] = hinfinorm(T,nt)

al1 = 0;

al2 = 10;

interval = al2-al1;

while interval>1e-5,

    gamma = (al1+al2)/2;

    truth = checknorm(gamma,T,nt);

    if truth == 0,

        al1 = gamma

    else

        al2 = gamma

    end

    interval = al2-al1

end

n = gamma;
```

7. function [hinf] = hinfnorm(t,nt)

```
[a,b,c,d]=systsep(t,nt);  
j=sqrt(-1);  
w=logspace(-1,3,200);  
w(1)=0;  
[row,col]=size(a);  
for i=1:1:200,  
    s=j*w(i);  
    g=c*inv(s*eye(row)-a)*b+d;  
    e=eig(sqrt(g'*g));  
    maxsing(i)=max(e);  
end  
hinf=max(maxsing);  
end
```

8. function [Tinv,ntinv] = invert(T,nt)

```
[at,bt,ct,dt] = systsep(T,nt);  
at1 = at-bt*inv(dt)*ct;  
bt1 = bt*inv(dt);  
ct1 = -inv(dt)*ct;  
dt1 = inv(dt);  
[Tinv,ntinv] = transfer(at1,bt1,ct1,dt1);
```

```
9. function [Tret,ntret] = invretdiff(T,nt)

[a,b,c,d]=systsep(T,nt);

[n,m]=size(d);

ieinv=inv(eye(n)+d);

at=a-b*ieinv*c;

bt=b*ieinv;

ct=-ieinv*c;

dt=ieinv;

[ar,br,cr,dr]=minreal(at,bt,ct,dt);

[Tret,ntret]=transfer(ar,br,cr,dr);
```

10. function [Tiplus,ntiplus] = iplus(gamma,T1,nt1)

```
[a,b,c,d]=systsep(T1,nt1);  
[row,col]=size(d);  
dd=gamma*eye(row)+d;  
[Tiplus,ntiplus]=transfer(a,b,c,dd);
```

11. function [dl,ndl,nl,nnl] = lcofac(tp,ntp,f,h)

```
[a,b,c,d]= systsep(tp,ntp);  
af=a-b*f;  
eaf=eig(af);  
cf=c-d*f;  
ah=a-h*c;  
bh=b-h*d;  
  
%Left coprime fractions of the plant  
[row,col]=size(c);  
[dl,ndl]=transfer(ah,h,-c,eye(row));  
[nl,nnl]=transfer(ah,bh,c,d);
```

12. function [vr,nvr,ur,nur] = lctrcofac(tp,ntp,f,h)

```
[a,b,c,d]= systsep(tp,ntp);  
af=a-b*f;  
eaf=eig(af);  
cf=c-d*f;  
ah=a-h*c;  
bh=b-h*d;  
  
%Left coprime fractions of the plant  
[row,col]=size(b);  
[vr,nvr]=transfer(ah,bh,f,eye(col));  
[rowh,colh]=size(h);  
[rowf,colf]=size(f);  
[ur,nur]=transfer(ah,h,f,zeros(rowf,colh));
```

```

13. function [f,h] = lqfdbk(tp,ntp)

[a,b,c,d]= systsep(tp,ntp);

q=eye(ntp);

[row,col]=size(b);

r=eye(col);

[f,s]=lqr(a,b,q,r);

%Stable system after lqr

af=a-b*f;

eaf=eig(af);

cf=c-d*f;

m=min(real(eaf));

%Placing the eigen values of ah to the left of eigen

%values of af.

i=1;

[row,col]=size(c);

r=eye(row);

[h,s]=lqr(a',c',i*q,r);

h=h';

ah = a-h*c;

```

```
eah=eig(ah);

n=max(real(eah));

while m <= n,
    i = i+1;
    if i > 10,
        break;
    end;

[h,s]=lqr(a',c',i*q,r);

h=h';
ah=a-h*c;

bh=b-h*d;

eah=eig(ah);

n=max(real(eah));

end
```

```
14. function [Tmul,ntmul] = mul(T1,nt1,T2,nt2)

    [At1,Bt1,Ct1,Dt1] = systsep(T1,nt1);
    [At2,Bt2,Ct2,Dt2] = systsep(T2,nt2);
    At = [At1 Bt1*Ct2;zeros(nt2,nt1) At2];
    Bt = [Bt1*Dt2;Bt2];
    Ct = [Ct1 Dt1*Ct2];
    Dt = [Dt1*Dt2];
    [at,bt,ct,dt] = minreal(At,Bt,Ct,Dt);
    [Tmul,ntmul] = transfer(at,bt,ct,dt);
```

```
15. function [Tmul,ntmul] = mull(T1,nt1,T2,nt2)

[At1,Bt1,Ct1,Dt1] = systsep(T1,nt1);

[At2,Bt2,Ct2,Dt2] = systsep(T2,nt2);

At = [At1 Bt1*Ct2;zeros(nt2,nt1) At2];

Bt = [Bt1*Dt2;Bt2];

Ct = [Ct1 Dt1*Ct2];

Dt = [Dt1*Dt2];

[at,bt,ct,dt] = minreal(At,Bt,Ct,Dt,1e-4);

[Tmul,ntmul] = transfer(at,bt,ct,dt);
```

16. function [Tneg,ntneg] = negate(T1,nt1)

[a,b,c,d]=systsep(T1,nt1);

[Tneg,ntneg]=transfer(a,-b,c,-d);

17. function [Q,nq] = nehari(T1,nt1,T2,nt2,T3,nt3)

```
[Uo,nuo] = spectrdns(T2,nt2);
[IUo,niuo] = invert(Uo,nuo);
[Ui,nui] = mul(T2,nt2,IUo,niuo)
[Uit,nuit] = tilde(Ui,nui);
[UUt,nuut] = mul(Ui,nui,Uit,nuit);
[UUtm,nuutm] = negate(UUtm,nuut);
[Tt1,ntt1] = iplus(1,UUtm,nuutm)
[Y,ny] = mul(Tt1,ntt1,T1,nt1)
if ny==0,
    [a,b,c,d]=systsep(Y,ny)
    normy=sqrt(max(eig(d'*d)));
else
    normy = hinfnorm(Y,ny)
end
alpha1 = hinfnorm(T1,nt1)
interval = (alpha1-normy);
while interval > 0.001,
    gamma = (alpha1+normy)/2
    truth = 0;
    while truth == 0,
```

```

[Tt1,ntt1] = tilde(Y,ny);

[Tt2,ntt2] = mul(Tt1,ntt1,Y,ny);

[Tt1,ntt1] = negate(Tt2,ntt2);

gammasq = gamma*gamma

[Tt2,ntt2] = iplus(gammasq,Tt1,ntt1);

[Yo,nyo] = spectral(Tt2,ntt2);

[Tt3,ntt3] = invert(Yo,nyo);

[Tt2,ntt2] = mul(T3,nt3,Tt3,ntt3);

[Tt1,ntt1] = transpose(Tt2,ntt2)

[Vcot,nvcot] = spectrdns(Tt1,ntt1);

[Tt2,ntt2] = invert(Vcot,nvcot);

[Vcit,nvcit] = mul(Tt1,ntt1,Tt2,ntt2);

[Vco,nvco] = transpose(Vcot,nvcot);

[Vci,nvci] = transpose(Vcit,nvcit);

[Tt1,ntt1] = tilde(Vci,nvci);

[Tt2,ntt2] = mul(Tt1,ntt1,Vci,nvci);

[Tt1,ntt1] = negate(Tt2,ntt2);

[Tt2,ntt2] = iplus(1,Tt1,ntt1);

[Tt1,ntt1] = mul(T1,nt1,Tt3,ntt3);

[Tt4,ntt4] = mul(Uit,nuit,Tt1,ntt1);

[Z,nz] = mul1(Tt4,ntt4,Tt2,ntt2);

```

```

truth=checknorm(1,Z,nz);

if truth == 0,
    gamma = (gamma+alpha1)/2;,

end;

end;

[Zt,nzt] = tilde(Z,nz);

[Tt1,ntt1] = mul(Z,nz,Zt,nzt);

[Tt2,ntt2] = negate(Tt1,ntt1);

[Tt1,ntt1] = iplus(1,Tt2,ntt2);

[Tt2,ntt2] = transpose(Tt1,ntt1);

[Zcot,nzcot] = spectral(Tt2,ntt2);

[Zco,nzco] = transpose(Zcot,nzcot);

[Zcoi,nzcoi] = invert(Zco,nzco);

[Tt1,ntt1] = muli(Zcoi,nzcoi,Tt4,ntt4);

[Vcitol,nvcitol] = tilde(Vci,nvci);

[R,nr] = muli(Tt1,ntt1,Vcitol,nvcitol);

hankelr = hanksingval(R,nr)

if hankelr < 0.99999,
    alpha1 = gamma;,

else

normy = gamma;;

```

```

end;

interval = alphai-normy

end;

gamma=alpha1

[Tt1,ntt1] = tilde(Y,ny);

[Tt2,ntt2] = mul(Tt1,ntt1,Y,ny);

[Tt1,ntt1] = negate(Tt2,ntt2);

gammasq = gamma*gamma;

[Tt2,ntt2] = iplus(gammasq,Tt1,ntt1);

[Yo,nyo] = spectral(Tt2,ntt2)

[Tt3,ntt3] = invert(Yo,nyo);

[Tt2,ntt2] = mul(T3,nt3,Tt3,ntt3);

[Tt1,ntt1] = transpose(Tt2,ntt2);

[Vcot,nvcot] = spectrdns(Tt1,ntt1);

[Tt2,ntt2] = invert(Vcot,nvcot);

[Vcit,nvcit] = mul(Tt1,ntt1,Tt2,ntt2);

[Vco,nvco] = transpose(Vcot,nvcot);

[Vci,nvci] = transpose(Vcit,nvcit);

[Tt1,ntt1] = tilde(Vci,nvci);

[Tt2,ntt2] = mul(Tt1,ntt1,Vci,nvci);

[Tt1,ntt1] = negate(Tt2,ntt2);

```

```

[Tt2,ntt2] = iplus(1,Tt1,ntt1);

[Tt1,ntt1] = mul(Tt1,ntt1,Tt3,ntt3);

[Tt4,ntt4] = mul(Uit,nuit,Tt1,ntt1);

[Z,nz] = mul1(Tt4,ntt4,Tt2,ntt2);

[Zt,nzt] = tilde(Z,nz);

[Tt1,ntt1] = mul(Z,nz,Zt,nzt);

[Tt2,ntt2] = negate(Tt1,ntt1);

[Tt1,ntt1] = iplus(1,Tt2,ntt2);

[Tt2,ntt2] = transpose(Tt1,ntt1);

[Zcot,nzcot] = spectral(Tt2,ntt2);

[Zco,nzco] = transpose(Zcot,nzcot);

[Zcoi,nzcoi] = invert(Zco,nzco);

[Tt1,ntt1] = mul1(Zcoi,nzcoi,Tt4,ntt4);

[Vcitol,nvcitol] = tilde(Vci,nvci);

[R, nr] = mul1(Tt1,ntt1,Vcitol,nvcitol)

hankelr = hanksingval(R, nr)

[s1,ns1,s2,ns2] = parts(R, nr);

[x,nx] = smallgain(s2,ns2)

[Tt1,ntt1] = mul1(Zcoi,nzcoi,Uo,nuo);

[Tt2,ntt2] = invert(Tt1,ntt1);

[Tt1,ntt1] = invert(Vco,nvco);

```

```
[Tt3,ntt3] = mul1(Tt2,ntt2,x,nx);  
[Q,nq] = mul1(Tt3,ntt3,Tt1,ntt1);
```

```
18. function [Tpar,ntpar] = parallel(T1,nt1,T2,nt2)

[At1,Bt1,Ct1,Dt1] = systsep(T1,nt1);

[At2,Bt2,Ct2,Dt2] = systsep(T2,nt2);

At = [At1 zeros(nt1,nt2);zeros(nt2,nt1) At2];

Bt = [Bt1;Bt2];

Ct = [Ct1 Ct2];

Dt = [Dt1+Dt2];

[at,bt,ct,dt] = minreal(At,Bt,Ct,Dt);

[Tpar,ntpar] = transfer(at,bt,ct,dt);
```

19. function [t1,nt1,t2,nt2] = parts(t,nt)

```
[ts,nts,v]=sorteig(t,nt);
[a,b,c,d] = systsep(ts,nts);
n=nts;
k=n;
fl=0;
for j = 1:1:n,
    temp=real(a(j,j))+1e-8;
    if temp >= 0,
        temp = a(j,j);
        k=j;
        fl = 1;
        break
    end
end
if fl == 0,
    k = n+1;
end;
if k>1,
    a1=a(1:k-1,1:k-1);
    b1=b(1:k-1,:);
end
```

```
c1=c(:,1:k-1);

[mb,nb]=size(b1);

[mc,nc]=size(c1);

d1=zeros(mc,nb);

else

a1=0;

b1=0;

c1=0;

d1=0;

end;

if k<n+1,

a2=a(k:n,k:n);

b2=b(k:n,:);

c2=c(:,k:n);

[mb,nb]=size(b2);

[mc,nc]=size(c2);

d2=zeros(mc,nb);

else

a2=0;

b2=0;

c2=0;
```

```
d2=0;  
end;  
[t1,nt1]=transfer(a1,b1,c1,d1);  
[t2,nt2]=transfer(a2,b2,c2,d2);
```

20. function [dr,nr,nnr]=rcofac(tp,ntp,f,h)

```
[a,b,c,d]= systsep(tp,ntp);  
af=a-b*f;  
cf=c-d*f;  
ah=a-h*c;  
bh=b-h*d;  
  
%Right coprime fractions of the plant  
[row,col]=size(b);  
[dr,nr]=transfer(af,b,-f,eye(col));  
[nr,nnr]=transfer(af,b,cf,d);
```

21. function [vl,nvl,ul,nul] = rctrcofac(tp,ntp,f,h)

```
[a,b,c,d]= systsep(tp,ntp);  
  
af=a-b*f;  
  
eaf=eig(af);  
  
cf=c-d*f;  
  
ah=a-h*c;  
  
bh=b-h*d;  
  
%Left coprime fractions of the plant  
  
[row,col]=size(c);  
  
[vl,nvl]=transfer(af,h,cf,eye(row));  
  
[rowh,colh]=size(h);  
  
[rowf,colf]=size(f);  
  
[ul,nul]=transfer(af,h,f,zeros(rowf,colh));
```

```

22. function [x,nx] = smallgain(r,nr)

[a,b,c,d]=systsep(r,nr)

[lc,lo]=findlyap(r,nr);

temp=lo*lc;

[row,col]=size(temp);

n=inv(eye(row)-temp);

[row,col]=size(c);

l1=transfer(a,-lc*n*c',c,eye(row));

l2=transfer(a,n'*b,c,zeros(row));

[row,col]=size(b');

l3=transfer(-a',n*c',-b',zeros(row));

l4=transfer(-a',n*lo*b,b',eye(row));

[row,col]=size(a);

nl2=row;

nl4=nl2;

[t14,nt14]=invert(l4,nl4);

[tm,ntm]=mul1(l2,nl2,t14,nt14);

[tr,ntr]=negate(tm,ntm);

[atest,btest,ctest,dtest]=systsep(tr,ntr);

[x,nx]=parallel(r,nr,tr,ntr);

```

```

23. function [ts,nts,v] = sorteig(t,nt)

    [a,b,c,d]=systsep(t,nt);

    [v,diag]=eig(a);

    am=diag;

    n=nt;

    for i=1:1:n-1;,

        for j=n:-1:i+1;,

            if real(am(j,j)) <= real(am(j-1,j-1)),

                temp=am(j,j);

                am(j,j)=am(j-1,j-1);

                am(j-1,j-1)=temp;

                temp=v(:,j);

                v(:,j)=v(:,j-1);

                v(:,j-1)=temp;

            end

        end

    end

    a=inv(v)*a*v;

    b=inv(v)*b;

    c=c*v;

    [ts,nts]=transfer(a,b,c,d);

```

24. function [ts,nts] = spectral(t,nt)

```
[a,b,c,d]=systsep(t,nt);

[t1,nt1,t2,nt2]=parts(t,nt);

[a1,b1,c1,d1]=systsep(t1,nt1);

[a1,b1,c1,d1]=minreal(a1,b1,c1,d1);

[row,col]=size(a1);

a=[a1 zeros(row); zeros(row) -a1'];

b=[b1;-c1'];

c=[c1 b1'];

ax=a-b*inv(d)*c;

[rax,cax]=size(ax);

[t1,nt1]=transfer(ax,zeros(rax,1),zeros(1,rax),0);

[tax,ntax,v]=sorteig(t1,nt1);

ei=eig(ax);

k=0;

for i=1:rax;,

    if ei(i)<0,

        k=k+1;,

    end;

end;

vv=v(:,1:k);
```

```
v1=vv(1:k,:);  
v2=vv(k+1:rax,:);  
x=v2*inv(v1);  
as=a1;  
bs=b1;  
cs=inv(sqrtm(d))*(c1+b1'*x);  
ds=sqrt(d);  
[ts,nts]=transfer(as,bs,cs,ds);
```

25. function [ts,nts]=spectrdns(t,nt)

```
[a,b,c,d]=systsep(t,nt);

dbar=d'*d;

h11= a-b*inv(dbar)*d'*c;

h12= -b*inv(dbar)*b';

h21= -c'*c+c'*d*inv(dbar)*d'*c;

h22=-h11';

h=[h11 h12; h21 h22];

[rh,ch]=size(h);

[t1,nt1]=transfer(h,zeros(rh,1),zeros(1,rh),0);

[th,nth,v]=sorteig(t1,nt1);

ei=eig(h);

k=0;

for i=1:rh,,

    if ei(i) < 0,

        k=k+1;,

    end;

end;

vv=v(:,1:k);

vi=vv(1:k,:);

v2=vv(k+1:rh,:);
```

```
x=v2*inv(v1);  
  
as=a;  
  
bs=b;  
  
cs=inv(sqrtm(dbar))*(d'*c+b'*x);  
  
ds=sqrtm(dbar);  
  
[ts,nts]=transfer(as,bs,cs,ds);
```

26. function [A,B,C,D] = systsep(T,nt)

```
[rowt,colt] = size(T);  
  
A = T(1:nt,1:nt);  
  
B = T(1:nt,nt+1:colt);  
  
C = T(nt+1:rowt,1:nt);  
  
D = T(nt+1:rowt,nt+1:colt);
```

```
27. function [Ttall,nttall] = tall(T1,nt1,T2,nt2)

    [At1,Bt1,Ct1,Dt1] = systsep(T1,nt1);

    [At2,Bt2,Ct2,Dt2] = systsep(T2,nt2);

    At = [At1 zeros(nt1,nt2);zeros(nt2,nt1) At2];

    Bt = [Bt1;Bt2];

    [r1,c1]=size(Ct1);

    [r2,c2]=size(Ct2);

    Ct = [Ct1 zeros(r1,c2);zeros(r2,c1) Ct2];

    Dt = [Dt1;Dt2];

    [at,bt,ct,dt] = minreal(At,Bt,Ct,Dt);

    [Ttall,nttall] = transfer(at,bt,ct,dt);
```

```
28. function [Ttilde,ntilde] = tilde(T,nt)

[At,Bt,Ct,Dt] = systsep(T,nt);

ntilde = nt;

At1 = -At';

Bt1 = -Ct';

Ct1 = Bt';

Dt1 = Dt';

Ttilde = transfer(At1,Bt1,Ct1,Dt1);
```

29. function [T,nt] = transfer(A,B,C,D)

nt = sqrt(prod(size(A)));

T = [A B;C D];

30. function [Ttrans,nttrans] = transpose(T,nt)

```
[At,Bt,Ct,Dt] = systsep(T,nt);  
nttrans = nt;  
At1 = At';  
Bt1 = Ct';  
Ct1 = Bt';  
Dt1 = Dt';  
Ttrans = transfer(At1,Bt1,Ct1,Dt1);
```

7 Examples

```
% This is a program to test the example in Francis Pg. 96

%
row=3;

col=1;

[t,nt]=convtf2ss(row,col);

[go,ngo]=spectrdns(t,nt);

[a,b,c,d]=systsep(go,ngo);

[num1,den1]=ss2tf(a,b,c,d,1);

roots(num1)'

roots(den1)'

%
% Check from Pg 97. Roots of num are -0.9837 and -0.2861
%
% and roots of den are -1 and -2.

%
[goi,ngoi]=invert(go,ngo);

[gi,ngi]=mul(t,nt,goi,ngoi);

[a,b,c,d]=systsep(gi,ngi);

[num1,den1]=ss2tf(a,b,c,d,1);

for i=1:3

roots(num1(i,:))'
```

```
end  
roots(den1),  
%  
% Check Pg 97. Roots of num1 are -2  
% Roots of num2 are 0, -1  
% Roots of num3 are 1, -2  
% Roots of den are -0.9837, -0.2861
```

ex1pg96

Make sure that you form a common denominator

Input the numerator coeffs. matrix for

1

input

num =

0 1 2

10 10 0

1 1 -2

Input the denom. vector

den =

1 3 2

t =

-3	-2	1
1	0	0
1	2	0
-20	-20	10
-2	-4	1

nt =

2

0 states removed

ans =

-0.9837 -0.2861

ans =

-2 -1

2 states removed

ans =

-2.0000

ans =

-1.0000 0.0000

ans =

-2.0000 1.0000

ans =

-0.9837 -0.2861

quit

4718 flops.

% This part tests Step 5 on Pg 109 of Francis.

%

a=[2.189 0;0 .4569];

b=[1.038;-0.9291];

c=[1 1];

d=0;

[r,nr]=transfer(a,b,c,d);

[x,nx]=smallgain(r,nr)

a =

2.1890 0

0 0.4569

b =

1.0380

-0.9291

c =

1 1

d =

0

0 states removed

4 states removed

x =

-1.1451	-0.8869	1.2365
3.5103	-3.5665	3.7711
0.0000	0.4730	0

nx =

2

[a,b,c,d]=systsep(x,nx)

a =

-1.1451 -0.8869

3.5103 -3.5665

b =

1.2365

3.7711

c =

0.0000 0.4730

d =

0

eig(a)

%

% Two roots of the den of our X.

%

ans =

-2.3558 + 1.2835i

-2.3558 - 1.2835i

%

% The four roots of the num of X on Pg. 109 are given below.

%

roots([1 2.656 1.033])'

```
ans =
```

```
-2.1827 -0.4733
```

```
roots([1 sqrt(7) 1])'
```

```
ans =
```

```
-2.1889 -0.4569
```

```
% The roots of the X obtained on Pg 109 are not the same
```

```
% as those above.
```

```
% This shows that the X is not unique.
```

```
%
```

```
% The four roots of den. of X on Pg. 109 are -3.108 and those below.
```

```
roots([1 2.67 1.081 1.075])'
```

```
ans =
```

```
-2.4064 -0.1318 - 0.6553i -0.1318 + 0.6553i
```

quit

12784 flops.

```

%
% This program works out the example on Pg 125 of Francis.

%
row=2;

col=2;

[t,nt]=convtf2ss(row,col);

[x,nx]=smallgain(t,nt)

[a,b,c,d]=systsep(x,nx);

[num1,den1]=ss2tf(a,b,c,d,1);

[num2,den2]=ss2tf(a,b,c,d,2);

for i=1:2

    roots(num1(i,:))'

end

%
% Check Pg. 127 Roots of num of X11 are -2.473, -0.6030 +/- j*0.1775

%
% Roots of num of X21 are -1.086, -0.5590 +/- j*0.6321

%
for i=1:2

    roots(num2(i,:))'

end

%
```

```
% Check Pg. 127 Roots of num of X12 are -1.054, -1.0, -1.0  
%  
% Roots of num of X21 are -1.101, -0.6395 +/- j*0.7033  
  
%  
  
roots(den1)',  
  
%  
  
% Check Pg. 127 Roots of den of X are  
% -84.53, -1.101, -0.6905 +/- j*0.7957  
  
%
```

ex1pg125

Make sure that you form a common denominator

Input the numerator coeffs. matrix for

1

input

num =

0.3906 -0.3906 0.3906

0 0.7813 -0.7813

Input the denom. vector

den =

1 -2 2 -1

t =

2.0000	-2.0000	1.0000	1.0000
1.0000	0	0	0
0	1.0000	0	0
0.3906	-0.3906	0.3906	0
0	0.7813	-0.7813	0

nt =

3

Make sure that you form a common denominator

Input the numerator coeffs. matrix for

2

input

num =

0

1.5625

Input the denom. vector

den =

1 -1

t =

1.0000 1.0000

0 0

1.5625 0

nt =

1

0 states removed

0 states removed

a =

0.5556	0.7721	1.8127	1.4638
-0.8315	0.5159	-1.1618	-0.4654
0.0000	0.1929	1.4554	0.3204
0.0000	-0.3171	-0.7486	0.4732

b =

0.6667	-0.3333
-0.3563	0.1782
0.6536	0.4811
0.0364	-0.7910

c =

0.0000 0.0000 0.5780 0.3516

0.0000 0.0000 0.1064 -1.9107

d =

0 0

0 0

0 states removed

8 states removed

x =

-0.5004 -18.0165 2.3557 22.5969 11.5582 -28.0289

0.8466	-0.0476	-0.0753	-0.5885	-0.3249	0.9539
-0.4614	49.5633	-7.5057	-61.5635	-31.6395	76.5735
0.3863	59.1929	-7.6958	-78.9584	-39.1368	95.8678
0.0000	0.0000	0.1099	0.0214	0	0
0.0000	0.0000	0.0029	-0.7971	0	0

nx =

4

ans =

-2.4725 -0.6032 - 0.1768i -0.6032 + 0.1768i

ans =

-1.0865 -0.5588 - 0.6323i -0.5588 + 0.6323i

ans =

-1.0536 -1.0000 -1.0000

ans =

-1.1008 -0.6397 - 0.7031i -0.6397 + 0.7031i

ans =

-84.5300 -1.1008 -0.6906 - 0.7957i -0.6906 + 0.7957i

quit

117932 flops.

